



Technische  
Universität  
Braunschweig



# Mathematische Methoden der Algorithmik – Vorlesung #06

Arne Schmidt

# Letzte Woche

# (Dualer) Simplex in Matrixform

Technischer Aufschrieb des Algorithmus.

Das lässt sich nun recht einfach implementieren.

Primal Simplex	Dual Simplex
<p>Suppose <math>x_{\mathcal{B}}^* \geq 0</math></p> <p>while (<math>z_{\mathcal{N}}^* \not\geq 0</math>) {</p> <p>    pick <math>j \in \{j \in \mathcal{N} : z_j^* &lt; 0\}</math></p> <p>    <math>\Delta x_{\mathcal{B}} = B^{-1} N e_j</math></p> <p>    <math>t = \left( \max_{i \in \mathcal{B}} \frac{\Delta x_i}{x_i^*} \right)^{-1}</math></p> <p>    pick <math>i \in \operatorname{argmax}_{i \in \mathcal{B}} \frac{\Delta x_i}{x_i^*}</math></p> <p>    <math>\Delta z_{\mathcal{N}} = -(B^{-1} N)^T e_i</math></p> <p>    <math>s = \frac{z_j^*}{\Delta z_j}</math></p> <p>    <math>x_j^* \leftarrow t</math></p> <p>    <math>x_{\mathcal{B}}^* \leftarrow x_{\mathcal{B}}^* - t \Delta x_{\mathcal{B}}</math></p> <p>    <math>z_i^* \leftarrow s</math></p> <p>    <math>z_{\mathcal{N}}^* \leftarrow z_{\mathcal{N}}^* - s \Delta z_{\mathcal{N}}</math></p> <p>    <math>\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}</math></p> <p>}</p>	<p>Suppose <math>z_{\mathcal{N}}^* \geq 0</math></p> <p>while (<math>x_{\mathcal{B}}^* \not\geq 0</math>) {</p> <p>    pick <math>i \in \{i \in \mathcal{B} : x_i^* &lt; 0\}</math></p> <p>    <math>\Delta z_{\mathcal{N}} = -(B^{-1} N)^T e_i</math></p> <p>    <math>s = \left( \max_{j \in \mathcal{N}} \frac{\Delta z_j}{z_j^*} \right)^{-1}</math></p> <p>    pick <math>j \in \operatorname{argmax}_{j \in \mathcal{N}} \frac{\Delta z_j}{z_j^*}</math></p> <p>    <math>\Delta x_{\mathcal{B}} = B^{-1} N e_j</math></p> <p>    <math>t = \frac{x_i^*}{\Delta x_i}</math></p> <p>    <math>x_j^* \leftarrow t</math></p> <p>    <math>x_{\mathcal{B}}^* \leftarrow x_{\mathcal{B}}^* - t \Delta x_{\mathcal{B}}</math></p> <p>    <math>z_i^* \leftarrow s</math></p> <p>    <math>z_{\mathcal{N}}^* \leftarrow z_{\mathcal{N}}^* - s \Delta z_{\mathcal{N}}</math></p> <p>    <math>\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}</math></p> <p>}</p>

# Implementierung des Simplex-Algorithmus



# Laufzeit

Das Dictionary hat  $m+1$  Zeilen und  $n+1$  Spalten.

In einer naiven Implementierung:

- Berechnung von  $j, t, i, s$  und die Berechnungen des Updates am Ende gehen in  $O(n + m)$ .
- Wie lange dauert das Berechnen von  $\Delta x_B$  bzw  $\Delta z_N$ ?
  - $B$  ist quadratisch mit  $m$  Zeilen und Spalten.
  - $B^{-1}$  ist quadratisch mit  $m$  Zeilen und Spalten.
    - Das müssen wir auch erst berechnen!
    - $\Rightarrow O(m^3)$  mit Gauss-Methode
  - $N$  ist eine  $m \times n$  Matrix.
  - $\Rightarrow O(m^3 + m^2n)$  Berechnungen.

Geht das besser?

Primal Simplex	Dual Simplex
Suppose $x_B^* \geq 0$ while ( $z_N^* \not\geq 0$ ) { pick $j \in \{j \in \mathcal{N} : z_j^* < 0\}$ $\Delta x_B = B^{-1} N e_j$ $t = \left( \max_{i \in \mathcal{B}} \frac{\Delta x_i}{x_i^*} \right)^{-1}$ pick $i \in \operatorname{argmax}_{i \in \mathcal{B}} \frac{\Delta x_i}{x_i^*}$ $\Delta z_N = -(B^{-1} N)^T e_i$ $s = \frac{z_j^*}{\Delta z_j}$ $x_j^* \leftarrow t$ $x_B^* \leftarrow x_B^* - t \Delta x_B$ $z_i^* \leftarrow s$ $z_N^* \leftarrow z_N^* - s \Delta z_N$ $\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}$ }	Suppose $z_N^* \geq 0$ while ( $x_B^* \not\geq 0$ ) { pick $i \in \{i \in \mathcal{B} : x_i^* < 0\}$ $\Delta z_N = -(B^{-1} N)^T e_i$ $s = \left( \max_{j \in \mathcal{N}} \frac{\Delta z_j}{z_j^*} \right)^{-1}$ pick $j \in \operatorname{argmax}_{j \in \mathcal{N}} \frac{\Delta z_j}{z_j^*}$ $\Delta x_B = B^{-1} N e_j$ $t = \frac{x_i^*}{\Delta x_i}$ $x_j^* \leftarrow t$ $x_B^* \leftarrow x_B^* - t \Delta x_B$ $z_i^* \leftarrow s$ $z_N^* \leftarrow z_N^* - s \Delta z_N$ $\mathcal{B} \leftarrow \mathcal{B} \setminus \{i\} \cup \{j\}$ }

# Exkurs: Inverse von Matrizen

Eine  $m \times m$ -Matrix  $B$  hat genau dann eine inverse  $m \times m$ -Matrix  $B^{-1}$  (also  $BB^{-1} = E$ , wobei  $E$  eine  $m \times m$  Einheitsmatrix ist), wenn  $\text{rg}(B) = m$ .

Algorithmus	Laufzeit
Gauss-Jordan	$O(n^3)$
Strassen	$O(n^{2.807})$
Coppersmith-Winograd (CW)	$O(n^{2.376})$
Divide & Conquer	$O(n^{\omega+o(1)}), 2 \leq \omega < 2.37134$

Allerdings ist das explizite Berechnen der Inversen nicht immer notwendig.

# Lösen von Subproblemen

$\Delta x_B = B^{-1}Ne_j = B^{-1}a_j$  ist die Lösung zu  $Bx = a_j$   
 $\Delta z_N = -N^T v$ , wobei  $v$  die Lösung zu  $B^T v = e_j$  ist.



# Lösen von Subproblemen

$\Delta x_B = B^{-1} N e_j = B^{-1} a_j$  ist die Lösung zu  $Bx = a_j$   
 $\Delta z_N = -N^T v$ , wobei  $v$  die Lösung zu  $B^T v = e_j$  ist.



Können wir  
die Matrix  $B$  so  
vorbearbeiten,  
dass...

... das Lösen  
des LGS  
schnell geht?

... die Datenstruktur bei  
einem Basiswechsel  
schnell aktualisiert  
werden kann?



# Idee: LU-Zerlegung

Faktorisiere die  $m \times m$ -Matrix  $B$  in zwei  $m \times m$ -Matrizen  $L$  und  $U$ , sodass:

$$L = \underbrace{\begin{pmatrix} 1 & 0 & \cdots & 0 \\ \ell_{2,1} & 1 & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \ell_{m,1} & \cdots & \ell_{m,m-1} & 1 \end{pmatrix}}_{\text{Untere Dreiecksmatrix}}, U = \underbrace{\begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,m} \\ 0 & u_{2,2} & \cdots & u_{2,m} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{m,m} \end{pmatrix}}_{\text{Obere Dreiecksmatrix}}$$

Beispiel: Bestimme die LU-Zerlegung von

$$A = \begin{pmatrix} 2 & 4 & 1 \\ 4 & 9 & 3 \\ 1 & 3 & 2 \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0.5 & 1 & 1 \end{pmatrix}, U = \begin{pmatrix} 2 & 4 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0.5 \end{pmatrix}$$

Wichtig:  $B^T = (LU)^T = U^T L^T$

# Berechnen von Gleichungssystemen mit LU

$\Delta x_B = B^{-1} N e_j = B^{-1} a_j$  ist die Lösung zu  $Bx = LUx = a_j$

Sei  $y := Ux$ , dann löse

- erst  $Ly = a_j$ ,
- dann  $Ux = y$

$\Delta z_N = -N^T v$ , wobei  $v$  die Lösung zu  $B^T v = (LU)^T v = U^T L^T v = e_j$  ist.

Sei  $y := L^T v$ , dann löse

- erst  $U^T y = e_j$ ,
- dann  $L^T v = y$

# Beispiel I – LU-Zerlegung

$$B = \begin{bmatrix} 2 & & 4 & & -2 \\ 3 & 1 & & 1 & \\ -1 & & -1 & & -2 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{bmatrix} = \overset{= L}{\begin{bmatrix} 2 & & & & \\ 3 & 1 & & & \\ -1 & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 7 \end{bmatrix}} \begin{bmatrix} 2 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 7 \end{bmatrix}^{-1} \overset{= U}{\begin{bmatrix} 2 & 4 & -2 & & \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix}}$$

Diese Matrizen können innerhalb einer einzigen Matrix gespeichert werden während der Gauß-Elimination (siehe Vanderbei Kapitel 8).

$$L = \begin{bmatrix} 2 & & & & \\ 3 & 1 & & & \\ -1 & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 7 \end{bmatrix} \begin{bmatrix} 2 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 7 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & & & & \\ \frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 1 \end{bmatrix}$$

# Beispiel II – Lösung von LGS über LU-Zerlegung I

$$a_j = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

Löse  $LU\Delta x_{\mathcal{B}} = a_j$  über  $Ly = a_j$  und  $U\Delta x_{\mathcal{B}} = y$

$$\begin{bmatrix} 1 & & & & \\ -\frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ -1 & -6 & & 1 & \\ 1 & & & & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ \frac{23}{2} \\ -\frac{7}{2} \end{bmatrix}$$

# Beispiel II – Lösung von LGS über LU-Zerlegung II

$$a_j = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix} \quad \text{Löse } LU\Delta x_{\mathcal{B}} = a_j \text{ über } Ly = a_j \text{ und } U\Delta x_{\mathcal{B}} = y \quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ \frac{23}{2} \\ -\frac{7}{2} \end{bmatrix}$$

$$\begin{bmatrix} 2 & & 4 & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ \frac{23}{2} \\ -\frac{7}{2} \end{bmatrix} \quad \Delta x_{\mathcal{B}} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 2 \\ 1 \\ -\frac{1}{2} \end{bmatrix}$$



# Sparsity

$$\begin{pmatrix} 1 & 4 & 0 & 1 & 0 \\ 2 & 1 & 2 & 3 & 4 \\ 0 & 4 & 1 & 2 & 1 \\ 2 & 5 & 8 & 3 & 2 \\ 9 & 5 & 0 & 5 & 1 \end{pmatrix}$$

„Voll“ (dense)

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

„Dünn“ (sparse)

Generell: Je mehr 0en eine Matrix besitzt, desto **dünnbesetzter** (sparser) ist die Matrix.

Dünnbesetzte Matrizen lassen sich einfacher und schneller in eine LU-Zerlegung teilen, und benötigen deutlich weniger Speicher.

# LU-Sparsity?



Können wir Spalten und Zeilen tauschen, um mehr Nullen in der LU-Zerlegung zu erhalten?

Dürfen wir das überhaupt?

Letztlich betrachten wir nur eine andere Reihenfolge von Variablen und Constraints!

# LU-Zerlegung – Komplexität

Generell ist das Ziel, die L bzw U Matrix so dünnbesetzt wie möglich zu halten, d.h. die Zahl der Nicht-Nullen („fill-ins“) soll so klein wie möglich sein.

**Problem:** Das Minimieren der Fill-Ins ist NP-schwer.

Es existieren aber gute Heuristiken, bspw.:

1. Tausche die Zeile mit den meisten nicht-eliminierten Nullen mit der aktuellen Pivot-Zeile
2. Tausche die Spalte mit den meisten nicht-eliminierten Nullen mit der Pivot-Spalte.

Beispiel:

$$\begin{pmatrix} 0 & 4 & 1 & 0 & 0 \\ 2 & 1 & 0 & 3 & 2 \\ 0 & 1 & 0 & 2 & 1 \\ 2 & 2 & 2 & 3 & 2 \\ 1 & 2 & 0 & 2 & 1 \end{pmatrix}$$

# Beispiel – LU-Heuristik

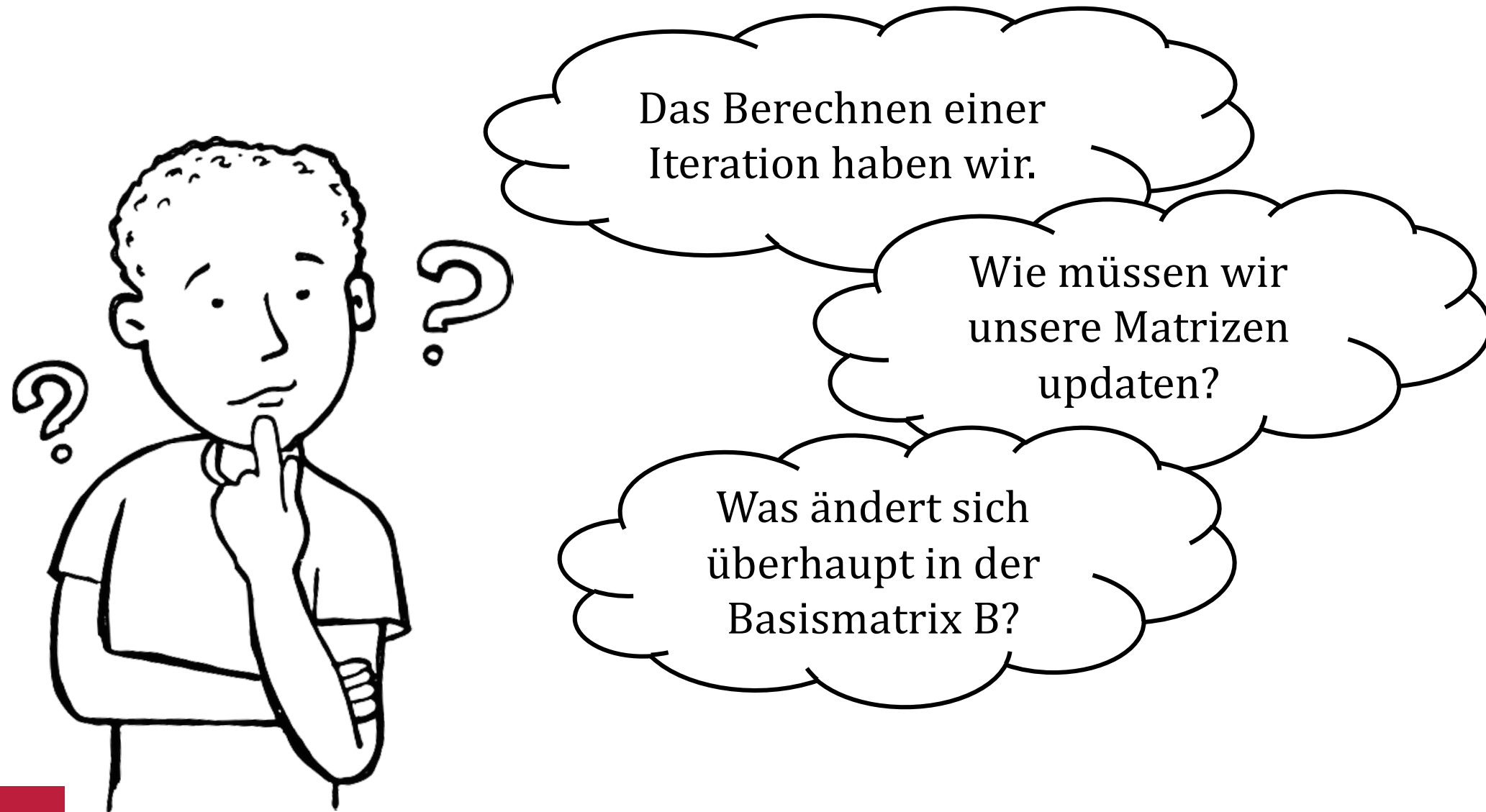
$$\begin{pmatrix} 2 & 1 & 0 & 3 & 2 \\ 0 & 4 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 \\ 2 & 2 & 2 & 0 & 2 \\ 1 & 2 & 0 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 4 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 2 \\ 0 & 1 & 0 & 2 & 0 \\ 2 & 2 & 2 & 0 & 2 \\ 0 & 2 & 1 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 \\ 0 & 3 & 2 & 1 & 2 \\ 2 & 0 & 2 & 2 & 2 \\ 0 & 0 & 1 & 2 & 1 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{3}{2} & 1 & 0 & 0 \\ 2 & -4 & 1 & 1 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{9}{26} & 1 \end{pmatrix},$$

$$U = \begin{pmatrix} 1 & 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 \\ 0 & 0 & 2 & -\frac{1}{2} & 2 \\ 0 & 0 & 0 & \frac{13}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

LU-Zerlegung besitzt 11 Nullen.  
Zerlegung über ursprüngliche  
Matrix hätte nur 8 Nullen.

# Aktualisieren nach Pivotschritt





# Basismatrix aktualisieren

Bei einem Pivotschritt (Basistausch) kann sich nur eine Zeile in  $B$  ändern, d.h.

$$B_{new} = B + (a_j - a_i)e_i^T = B \underbrace{\left( I + B^{-1}(a_j - a_i)e_i^T \right)}_{E :=}$$

Wir kennen dabei  $B^{-1}a_j = \Delta x_B$  und  $B^{-1}a_i = e_i$ . Also ist:

$$E = \left( I + (\Delta x_B - e_i)e_i^T \right)$$

Man kann nun zeigen, dass:

$$E^{-1} = I - \frac{(\Delta x_B - e_i)e_i^T}{\Delta x_i}$$

# Nächste Iteration

Für die nächste Iteration muss dann bspw. berechnet werden:

$$B_{new}\Delta\tilde{x}_B = BE\Delta\tilde{x}_B = \tilde{a}_j \text{ und } B_{new}^T\tilde{v} = E^TB^T\tilde{v} = \tilde{e}_i$$

Berechne diese wieder in Schritten

- $Bu = \tilde{a}_j$
- $E\Delta\tilde{x}_B = u \rightarrow \Delta\tilde{x}_B = E^{-1}u = u - \frac{u_i}{\Delta x_i}(\Delta x_B - e_i)$

Sowie

- $E^Ty = \tilde{e}_i \rightarrow y = (E^{-1})^T\tilde{e}_i = \tilde{e}_i - e_i \frac{(\Delta x_B - e_i)^T\tilde{e}_i}{\Delta x_i}$
- $B^T\tilde{v} = y$

# Viele Pivotschritte



Wenn wir  $k$  Iterationen durchgeführt haben, können wir den nächsten Schritt über die erste Basis berechnen

$$\text{Dann ist } B_{\text{new}} = B E_0 E_1 \dots E_{k-1}$$

Für  $B$  ist die LU-Zerlegung bekannt und die E-Matrizen sind sehr einfach.

Wenn wir  $B_{\text{new}}$  nicht explizit berechnen, dauert das ggf. irgendwann sehr lange.

# Überlegungen für viele Iterationen

Also, nach  $k$  Iterationen müssen wir folgende Informationen speichern:

- Basismatrix  $B$
- „Eta-Matrizen“  $E_0, \dots, E_{k-1}$  (dafür reicht es,  $\Delta x_B^j$  und einen Integer für die Spalte zu merken, die sogenannte eta-file)

Wann ist der Aufwand  $BE_0 \dots E_{k-1}$  zu berechnen zu hoch?

Wann lohnt es sich  $B_{new}$  in LU zu zerlegen?

Eine sorgfältige Analyse zeigt, dass nach etwa  $\sqrt{m}$  Iterationen eine neue Basismatrix explizit berechnet und zerlegt werden sollte.

In der Praxis ist die Anzahl dieser Iterationen sogar frei wählbar!

# Partial Pricing



# Viele Variablen, wenig Constraints



Viele Variablen  
erzeugen viel Arbeit:

Welches Pivot als  
nächstes?

Man muss die Länge  
des Pivotschritts  
berechnen.

Man muss  
reduzierte Kosten  
berechnen.

Müssen wir immer  
alle Variablen  
anschauen?

Interessant sind  
doch nur die „guten“  
Variablen.

# Partial Pricing

Idee: Betrachte zu jedem Zeitpunkt nur einen Teil der Variablen und aktualisiere das gesamte Dictionary nur ab und an.

Dazu:

- Suche unter den ersten  $\frac{n}{3}$  Variablen die 40 Variablen, welche den größten negativen Koeffizienten besitzen.
- Betrachte nur diese Variablen für die Iterationen bis nur noch ein Teil dieser (bspw, weniger als die Hälfte) für einen Pivotschritt in Frage kommen.
- Aktualisiere dann das gesamte Dictionary.

Damit müssen wir in jeder Iteration nur mit wenig Variablen arbeiten. Nur ab und an trifft uns die Härte der Realität.

# Steepest Edge

# Wir gehen steil!

Bisher oft nach größtem Koeffizienten ausgewählt. Das liefert die größte Änderungsrate der Zielfunktion und betrachtet nur die Nichtbasisvariablen.

Schließe die Basisvariablen mit ein! Da die Zielfunktion in der Form

$$f(x) = c_B^T x_B + c_N^T x_N$$

ist, ist die Ableitung von  $f(x)$  in Richtung  $\Delta x$  für ein  $j$ :

$$\frac{\partial f}{\partial \Delta x} = \frac{c_B^T \Delta x_B + c_N^T \Delta x_N}{\|\Delta x\|} = \frac{-z_j^*}{\sqrt{\|B^{-1} N e_j\|^2 + 1}}$$

Okay, der Nenner sieht nicht gut aus. Zu Beginn ist  $B$  aber quasi eine Einheitsmatrix. Dadurch können wir

$$v_k = \|B^{-1} N e_k\|^2 \text{ für alle } k \in \mathcal{N}$$

einfach berechnen.

# Aktualisiere $v_k$

Wir können herleiten, wie sich die  $v_k$  in der nächsten Iteration verändern.  
Dazu sei

$$\begin{aligned} u &:= (B^{-1})^T e_i \\ w &:= (B^{-1})^T \Delta x_{\mathcal{B}} \end{aligned}$$

$u$  berechnen wir sowieso schon.  $w$  muss zusätzlich berechnet werden. Damit können wir folgendes berechnen:

$$\tilde{v}_k = v_k - 2 \frac{a_k^T u (w - u)^T a_k}{\Delta x_i} + (a_k^T u)^2 \frac{\|\Delta x_{\mathcal{B}} - e_i\|^2}{(\Delta x_{\mathcal{B}})^2}$$

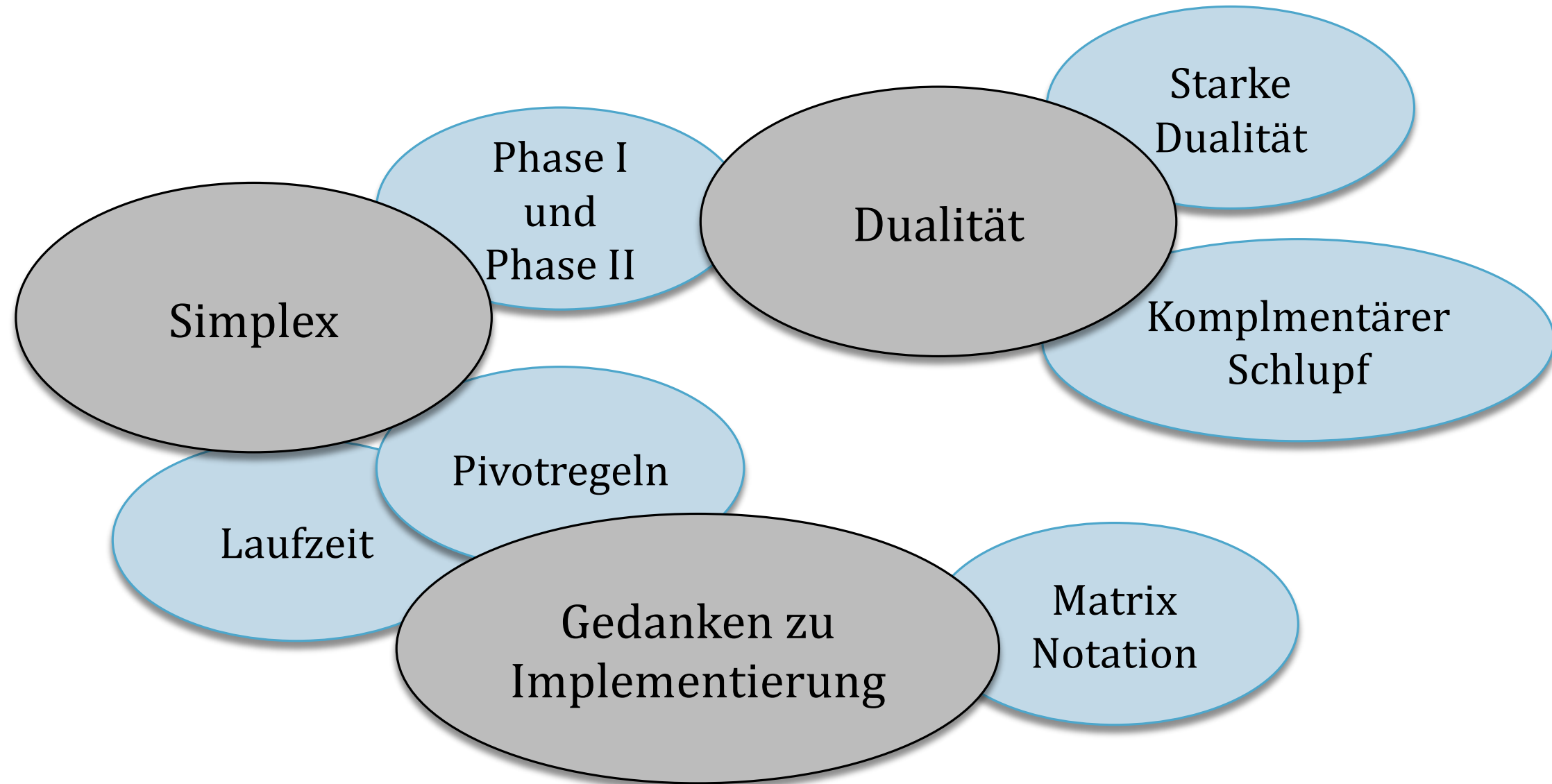
Wähle damit  $j \in \mathcal{N}$ , sodass  $\frac{-z_j^*}{\sqrt{v_j+1}}$  kleinstmöglich ist.

Die Steepest-Edge-Regel performt in der Praxis sehr gut mit den angegebenen Formeln.



# Recap

# Was bisher geschah...



# ... und was noch kommt



Bisher nur betrachtet:  $Ax \leq b$ .  
Was ist, wenn wir LPs der  
Form  $b_\ell \leq Ax \leq b_u$  haben?

Bisher nur reelle Lösungen  
angeschaut, aber viele Probleme  
fordern ganzzahlige Lösungen!

Wie unterschiedlich sind  
Probleme auf diskreten  
Strukturen (bspw. Graphen)  
oder in geometrischem Setting?